C# Coding Exercises

Practice 1: C#

Develop a program/method (console application) to do basic calculations.

Input: Your program accepts two inputs (value1, value2)

Input: You program accept options like 1(sum of two values), 2(subtraction of 2 values), 3(multiplication of two values)

Output: for the given two inputs, based on the option you choose it should produce the results

Example: input/output

Input: value1 = 10, value=20

Input option: 1

Output: 30

Practice 2: C#

Develop a program/method to generate the greeting (Console Application)

Input: Your program accepts your name(string)

Output: Your program should return Greeting message based on current time.

Example: input/output

Input: “Satya”

Output: “Hello Satya, Good Afternoon!” (it differs based on time)

Practice 3: C#

Develop a program/method to print sum of the digits of given number.

Practice 4: C#

Develop a program to build car factory

Instructions:

Create a Car Model (Id, Name, Model, Make, Year, Speed, Weels, Miles)

Create a Factory to build a car for given inputs

Design methods like InitializeCar, SetSpeed, SetModel, SetMake, SetYear, SetWeels, SetMiles, GetCarInfo

Inputs:

Name, Model, Make, Speed, Weels, Miles

Output:

Hello you choose {Name} {Make} of {Model} with {} weels with speed of {Speed} and it runs {Miles}

Note:

Extend this program to support any kind vehicle. (Car, Bike, Motor-Cycle)

Use Inheritance, Constructor, properties, Exception Handling

Practice 5: C#

Develop a program to generate banking transaction

Instructions:

Create a Customer Model class (CustomerId, CustomerName, Address, AccountBalance)

Create Bank truncation class

Initialize the Customer Object

Set the Initial balance

Develop a method in Bank class to Deposit amount to the customer

Develop a method in bank class to Withdraw amount from the customer

Develop a method to get the customer balance

Develop a method to check whether customer has enough balance or not

Develop a method to return customer info.

Input:

Customer (123, “Satya”, “Cranston, RI”, $500)

Deposit (500) – Balance becomes $1000

Withdraw (100) – Balance becomes $900

Withdraw(5000) – throw error insufficient balance

GetCustomerInfo() – 123, “Satya”, “Cransonton, RI”, {Balance}

Note:

Use properties, constructor, Exception Handling and List

Practice 6: C#

Develop a program to build Employee Management

Instructions:

Create an Enum (EmployeeType) – Permanent, Contractor, Intern

Create a JobTitle enum – Manager, Developer, Testor, Admin

Create an Employee model (Id, FirstName, LastName, EmployeeType (enum type), JobTitle(enum type), HourlyRate, Address, Email, Phone)

Create a HrStore class to manage employees

Initialize Employee Object in HrStore class

Develop methods to Add()/Remove()/Get()/Update () employees

Note:

Use properties, constructor, Exception Handling and List

Practice 7: C#

Create an Extension methods for below

- Create an extension method return first 4 letters of given string
- Create an extension method to return last 4 letters of given string
- Create an extension method return just string part of given email id without domain
- Create an extension method to return file extension for given file path
- Create an extension method return number of words in the given sentence.

Practice 8: C#

Write a program to do various operations on given collection using LINQ methods

Instructions:

Create a model of Employee (Id, Name, Job, City, Country, Salary)

Create a List of Employee model and initialize employee list

Build various operators on the employee list:

Get the list of employees whose city is “RI”
Get the list of employees whose salary is more than $5000
Get the list of employees whose Country is USA
Get the Count of employees
Get the employees whose JobTitle is “Software Engineer”
Get the employee City is “Johnston” and JobTitle is “Manager”

Practice 9: C# (using an interface, inheritance, Generics, collections, delegates)

Develop a program to build generic Repository to perform CRUD operations on various entities for a Store.

CRUD (CREATE, READ, UPDATE, DELETE)

Instructions:

Create an Entity for Customer (Id, Name, Address, Email, Phone, Account Balance)

Create an Entity for Employee (Id, Name, Address, Email, Phone, JobTitle, JobType, Salary)

Create a generic interface for CRUD operations

```
public interface IRepository<T>
{
 void Add<T>(T entity);
void Update<T>(T entity);
void Remove(T entity);
T Get(int id);
List<T> GetAll()
}
```

Implement IRepository

e.g.

```
public class CustomerRepository: IRepository<Customer>
{
    //build CRUD operations for Customor
}
public class EmployeeRepository: IRepository<Employee>
{
    //build CRUD operations for Employee
}
```

Build a store class to manage Customers and Employees

```
public class Store
{
    IRepository<Customor> custRepo = new CustomoerRepository();
    IRepository<Employee> empRepo = new EmployeeRepository();
    //do all the CRUD operations on both repos
}
```